# 1 Execution Service for Distributed Processors

**Takako M. Hickey, Caltech**

## 1.1 Technical Steps that went into arriving at the functional prototype

1. Downloaded Objective Caml compiler [4].
2. Downloaded Ensemble group communication software [2].
3. Developed the first version of the service in Caml over Ensemble.
4. Installed it on Linux Beowulf cluster nagegling.cacr.caltech.edu [1].
5. Tested it with a small set of ORCA jobs.
6. Found that the first version was unstable over 32 processors.
7. Redesigned and coded the second version.
8. Installed the second version on naegling. It appears stable with 64 processors.

## 1.2 Reasons for taking the technical steps described in Previous Section

The ultimate goal of the project is to develop a tool that aids high energy physicists in effectively using computational resources distributed worldwide. There are a number of tools available today that are aimed toward similar goals (LSF, PBS, DQS, Condor etc.), but none quite adequate. The first step in the project was to take a look at a few technical issues that are important to high energy physics experiments that have not been addressed by existing tools. These are:

1. keeping track of large number of long running jobs
2. supporting collaboration among multiple physicists
3. conserving limited network bandwidth
4. maintaining high availability
5. tolerating partition failures that are common in wide-area networks

Our execution service addresses issues a and b by introducing the notion of sessions. A session serves as a container for multiple jobs that are related. Jobs that are in the same session can be examined or killed with a single command. A session can be shared by multiple physicists, and safe access to it are ensured by our service.

Compared to the amount of data high energy physics experiments produce, today's network is slow. Our execution service addresses this problem by allowing processors to be selected based not only on load or type but also on where data resides.

High availability of service is addressed by having multiple copies of servers. Servers share some state so that resources will not be over-allocated or under-utilized. The shared state is replicated at each server and its consistency is maintained. It turns out that in the environment where network partitions can occur availability and consistency are at odds with each other.

Implementation of our execution service utilizes two existing technologies: functional language ML and group communication toolkit. ML is strongly typed and does garbage collection. These two features makes programs written in ML much less prone to bugs compared to those written in more popular languages such as C/C++ and Java. Group communication aims to facilitate programmers in writing distributed programs. Two important concepts are group membership and ordered communication. Group membership keeps track of members that are reasonably communicable. Ordered communication provides delivery of messages within each group that guarantees specified properties. An example is totally ordered communication, which delivers the same set of messages during each view (an instantiation of membership) in the same order. These properties are useful in maintaining replicated state consistent. Both ML and group communication are founded on rigorous theory. They are still active area of research in computer science community and tools we selected are at the cutting edge of technology that computer science has to offer.

While group communication facilitates in writing fault-tolerant distributed programs, it is not a panacea. One of the problems is its inadequacy in dealing with partition failures. When a new member joins a group the default action group communication performs is to transfer the group state to the newly joined member. The problem with this method is that the newly joined member may not be a newly started member but an old member that has partitioned away. This is possible because in distributed environment one cannot distinguish a process that is faulty from a process that is simply slow, and removing members that are too slow from a group is sometimes necessary in order to make progress. When a partitioned member rejoins the group and receives the group state, the old state that was held by the partioned member will be overwriteen. In a practical term, this could mean a loss of an access to an active computation. Our execution service gets around this problem by merging state on membership changes rather than relying on the simple state transfer. For more detail description for the design of our execution service, see [3].

Another problem with group communication is that because of its semantically rich operation, it does not scale to large number of members. While we knew of the existence of this problem, we run into the problem much earlier than we expected. Our initial remedy was to restrict the use of group communication to a small set of servers while connecting most of other members through cheaper form of communication.

## 1.3 Future Plan

The priority is placed on minimum features required for a system usable for CMS applications. Features nice to have but not necessary are placed under the time permitting section. Items are grouped in three month blocks that may be interchanged based on processor availability and people's schedules.

**Dec 2000** (or when more processors become available; with Vladmir Livtin)

- More testing: larger number of processors, more complicated ORCA scenarios.

**Mar 2000**

- Support for multiple users. Run server as root and run jobs under real user id.
- Security.
- Queuing when resources are not available.

**Jun 2001** (with Koen Holtman and Asad Samar)

- Integration with data replication service.
- Hook to Globus.

**Sep 2001** (with Iosif Legrand)

- Study of data aware scheduling algorithms using Monarc simulator.
- Integration with SONN agent.

**Dec 2001**

- Hirerachical servers for scalability.
- Performance monitoring: collect statistics on how long it takes to copy certain data etc. to better estimate scheduling.

**March 2002**

- Larger scale tests involving joint scheduling, and load balancing among the the interactive and production services at FNAL, CERN and Tier2 sites.
- Tests of joint scheduling at multiple CMS sites and shared-resource sites (e.g. NPACI sites in the US) for CPU-intensive.

**June 2002**

- Prepare multisite scheduling and load balancing tests for the Software and Computing Technical Design Report (TDR).

**December 2002**

- Large scale tests among existing Tier1 and Tier2 prototypes in the US and other CMS sites, in preparation for the Physics TDR in 2003.

**Time permitting**

- Multiple level queues.
- Better user interface following examples of more developed cluster computing systems such as PBS and DQS.
- More detailed study of scalability of current design, perhaps using the network simulator ns [5].

# References


[1] CACR. Intel pentium pro beowulf cluster naegling. http://www.cacr.caltech.edu/resources/naegling.

[2] The Ensemble project. The Ensemble distributed communication system. http://www.cs.cornell.edu/Info/Projects/Ensemble/index.html.

[3] Takako M. Hickey and Robbert van Renesse. An execution service for a partitionable low bandwidth network. In *Proceedings of the Twenty-Ninth International Symposium on Fault-Tolerant Computing*, Madison, Wisconsin, USA, June 1999.

[4] Xavier Leroy. The Objective Caml ssystem. http://pauillac.inria.fr/ocaml/.

[5] NS project. The network simulator: ns-2. http://www.isi.edu/nsnam.ns.